EECS 280 - Lecture 4

Arrays

https://eecs280staff.github.io/notes/04_Arrays.html

Announcements

- P1 due tonight at **8 pm** (**NOT** midnight)
- Remember to fill stuff out!
  - CARES survey (0.5% of your total grade) by 1/26
  - Coaching form (Piazza @105)
  - Exam accommodations form
  - Exam conflict forms 1/28
- No labs this week
  - Setup clinics if you want extra help getting your tools setup
- My OH today are switched to 2-3 and 4:30-5 (see calendar)

Clarification on Grading

- “What do I need to do to pass?”
  - 70% exam average (after curve) **AND**
  - 60% project average
- If you’re below either, you can’t get above a C-
- Otherwise you get this grade ->

Overall %	Grade
0 - 50%	E
50 - 60%	D
60 - 70%	C-
70 - 77%	C
77 - 80%	C+
80 - 83%	B-
83 - 87%	B
87 - 90%	B+
90 - 93%	A-
93 - 97%	A
97 - 100%	A+

Last Time

- Testing (how to do it effectively)
- Pointers (what they are, how to use them)
  - '*' and '&' can be used for type declarations
    - pointers and references, respectively
  - OR they can be used as operators to translate between objects and pointers

Lingering Questions

- “For the last RME example I am still feeling a little unclear. If you were to use an empty Vector and nothing happens couldn't that be considered a bug because the function isn't working as intended (even if the code executes without issue)?”
  - REQUIRES is usually used to describe what is needed for predictable behavior, even if that behavior is “nothing”

```
// REQUIRES: ???
//
// MODIFIES: ???
//
// EFFECTS:  ???
//
void mystery(vector<int> &v) {
  for (int i = 0; i < v.size(); i++) {
    v[i] = v[i] / v.size();
  }
}
```

Agenda

- **Finish up pointers**
- Arrays
  - Array decay
- Discussion on recent university news

Pass by Pointer

What should we pass into "swap_p"?

A) a, b
B) *a, *b
C) &a, &b

- Pointers give us an alternative to "passing by reference"

```
void swap(int& x,int& y) {
  int temp = x;
  x = y;
  y = temp;
}

int main() {
  int a = 3;
  int b = 7;
  swap(a, b);
}
```

```
void swap_p(int* x,int* y) {
  int temp = *x;
  *x = *y;
  *y = temp;
}

int main() {
  int a = 3;
  int b = 7;
  swap_p(  ??  );
}
```

Pointer Details

- There is a separate pointer type for each kind of thing you could point to, and you can't mix them.

```
int main() {
  int x = 3;
  double y = 4;
  int* ptr1 = &x;
  double* ptr2 = &y;
  ptr2 = &x; // compiler error!
}
```

Using Pointers in Expressions

- Dealing with pointer expressions can be confusing
- It can be helpful to keep track of what data type each object is
  - Using ‘&’ next to an object yields an object with an extra ‘*’ in the data type
  - Using ‘*’ next to a pointer object yields an object with one fewer ‘*’ in the data type

```
int  x = 3;

int* p;
p = &x;

int  y;
y = *p;
```

Why Pointers?

- Main tool which lets you do more in 280 vs. 183/101

- Where we'll get to: can create places in memory with no name, need to use pointers

Note on Notation

- I've been declaring pointer types this way

```
int* ptr1 = &x; // space after *
```

- But you can also do it this way

```
int *ptr1 = &x; // space before *
```

- The second one is more common in practice

Null and Uninitialized Pointers

- A null pointer has value 0x0 (i.e. it points to address 0)
  - No objects are allowed to live at address 0.
  - A null pointer is interpreted as "not pointing to anything".
  - Dereference a null pointer → undefined behavior (usually a runtime error).

```
int main() {
  int *ptr = nullptr;
  cout << ptr << endl;
  // prints 0
  cout << *ptr << endl;
  // probably crashes
}
```

Exercise: Pointers 4

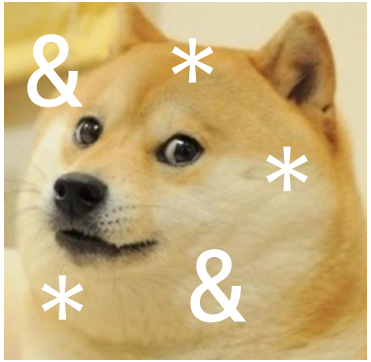
- Find the file “L03.4_pointer_mischief” on Lobster.

```
int * getAddress(int x) {
  return &x; // It's a trap!
}
void printAnInt(int someInt) {
  cout << someInt << endl;
}
int main() {
  int a = 3;
  int *ptr = getAddress(a);
  printAnInt(42);
  cout << *ptr << endl;
}
```

Why is it a trap?

A) Can't return pointers from functions
B) someInt became a reference to x
C) The lifetime of the parameter x ended before ptr was used
D) ptr became uninitialized when printAnInt was called

So Many * and &



- Used to specify a type…
  - * means it's a pointer
  - & means it's a reference

```
int* ptr;
```

```
int& ref;
```

- Used as an operator in an expression…
  - * means get object at an address

```
cout << *ptr << endl;
```

  - & means take address of an object

```
cout << &x << endl;
```

References vs. Pointers

References	Pointers
int& x	int* x
An alias for an object	Stores address of an object
Cannot rebind to another object	**Can** change where it points
Cannot refer to NULL (safer)	**Can** point to NULL (trickier)

```
int main() {
  int x = 3;
  int& y =  x;
  int* z = &x;
}
```

What can you do with pointers?

- Work with objects indirectly.
  - “Simulate” reference semantics.
  - Use objects across different scopes.
  - Enable subtype polymorphism.[1]
  - Keep track of objects in dynamic memory.[1]

[1] We’ll look at these later in the course.

Agenda

- Finish up pointers
- **Arrays**
  - Array decay
- Discussion on recent university news

Arrays in C++

- In C++ an array is a very simple *collection* of objects.
- Arrays…
  - …have a fixed size.
  - …hold elements of all the same type.
  - …have ordered elements.
  - …occupy a *contiguous* chunk of memory.
  - …support constant time random access. (i.e. “indexing”)

Arrays in C++

- Arrays are declared by modifying the base type using the “[]” notation, with the size listed inside the brackets:

```
int array_name[4]; // array of 4 ints
```

- Specific elements are also accessed using “[]”

```
cout << array_name[0]; // prints 0th int
```

Arrays in C++

- Array indexing always starts at 0
- Size of array (when specified), needs to either by hardcoded, or set by a “const” variable
  - We’ll discuss how to get around this limitation later

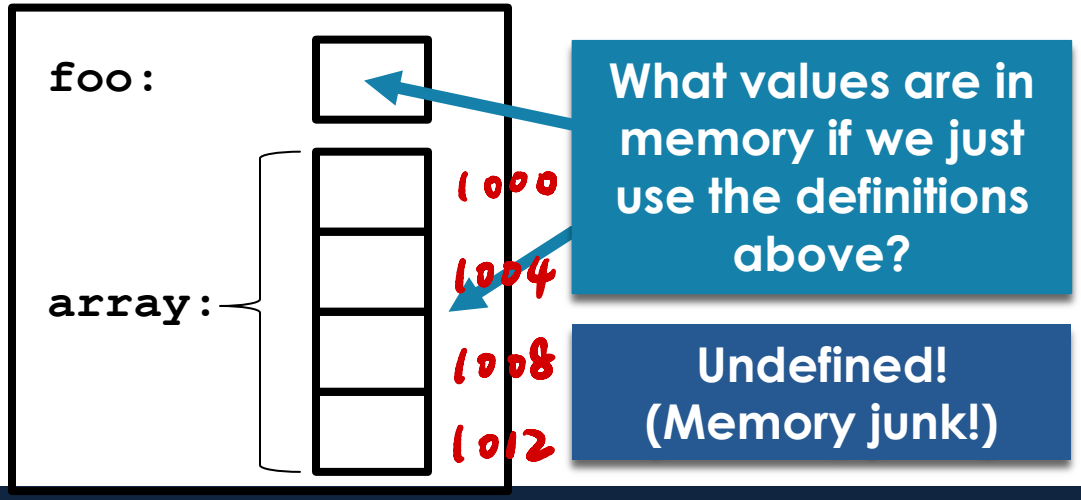
```
const int arr_size = 4;
int array_name[arr_size]; // ok
cin >> arr_size_2;
int another_array[arr_size_2]; // bad!
```

before run it, the compiler can figure out how much space to allocate for that array.

Example: Creating Arrays

- For comparison purposes, let's also declare and define an integer, `foo`:

```
int foo;
int array[4];
```

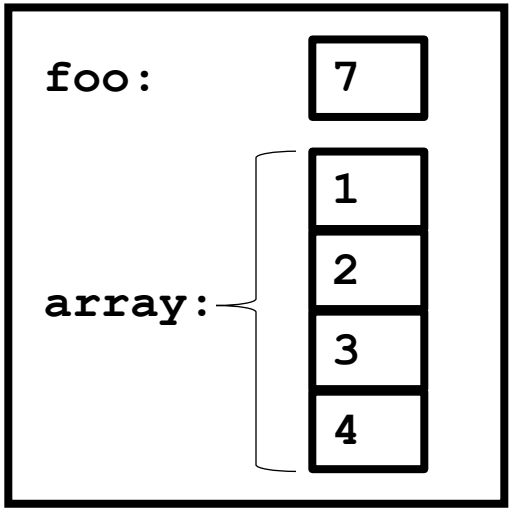


Example: Creating Arrays

- You can also initialize the contents of an array in one line – just like with an `int`. However, we need some sort of notation to specify a set of numbers:

```
int foo = 7;
int array[4] = { 1, 2, 3, 4 };
```

This is called an "initializer list".

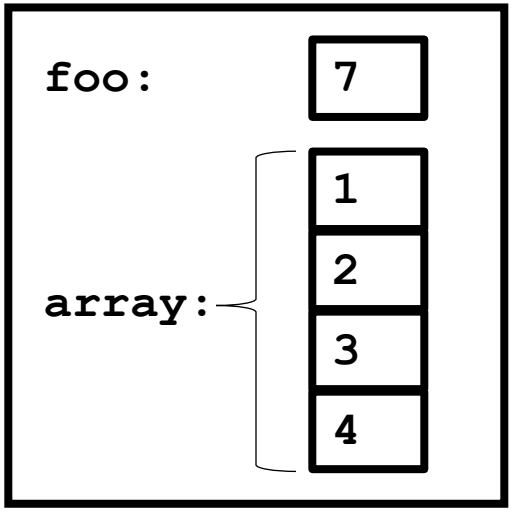


Example: Creating Arrays

- Array size can be omitted when using initializer lists:

```
int foo = 7;
int array[]  = { 1, 2, 3, 4 };
```

Compiler infers this has size four



Array Decay

- If you try to read the "value" of the whole array (instead of one of its elements)…
  - It **decays** into a **pointer to its first element**
  - Doesn't apply if you use "*" or "[]"
  - Doesn't apply if array appears to left of "="

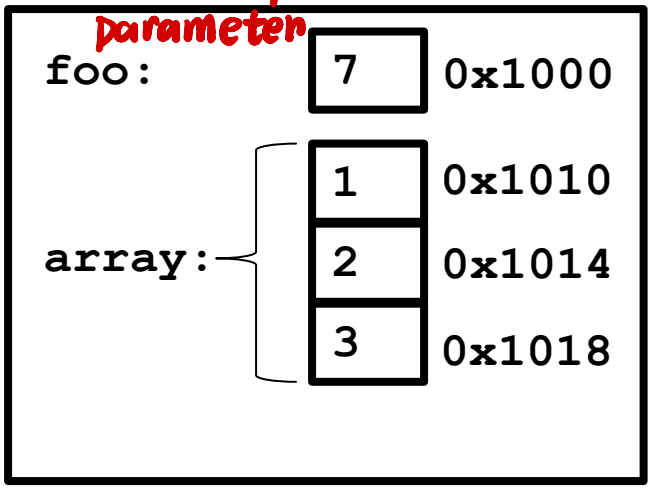
```
int foo = 7;
int array[3] = { 1, 2, 3 };
cout << array << endl;
```

Prints "0x1010"!

Poll: Which of the following compiles and results in pointer decay?

```
A)  cout << array[0];
B)  array + 1;
C)  array = &foo;
D)  cout << array;
E)  func(array);
```

Pass a pointer as a parameter

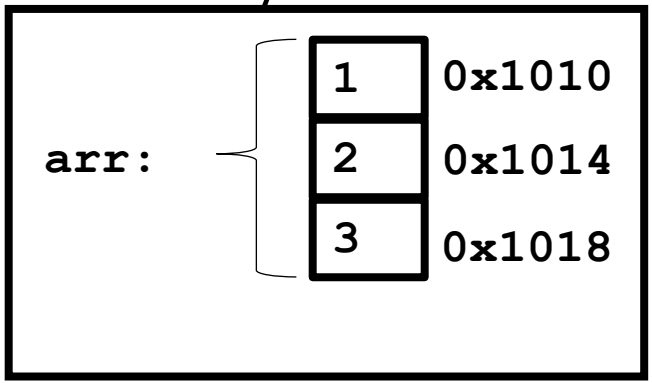


Pointer Arithmetic

- Array decay also let's us do **pointer arithmetic** to access array elements
  - Meaning: we use operands like "+" and "-" on pointer values to return new pointer values
  - An alternative (and equivalent) way to access array elements

```
int arr[3] = { 1, 2, 3 };
int *ptr = arr;
int *ptr2 = &arr[2];
int *ptr3 = arr + 2;
// ptr3 == ptr2
```

These statements are identical



Pointer Arithmetic

- How does pointer arithmetic work?
  - `int *ptr;` The compiler knows how big an `int` is. (let's say 4 bytes)
  - `ptr + x` computes the address `x` `int`s forward in memory
  - Operators: `+`, `-`, `+=`, `-=`, `++`, `--`
- Warning! Pointer arithmetic only makes sense in arrays!
  - Arrays are guaranteed to be **contiguous** memory.

```
int x = 42;
int arr[5] = { 1, 2, 3, 4, 5 };

// What's 2 spaces past the first element of arr? Easy.
int *goodPtr = arr + 2;

// What's 2 spaces past x? Could be anything!.
int *badPtr = &x + 2;
```

Memory diagram

Pointer Arithmetic

- We can also use comparison operators with pointers.

  <, <=, >, >=, ==, !=

- These just compare the address values numerically.

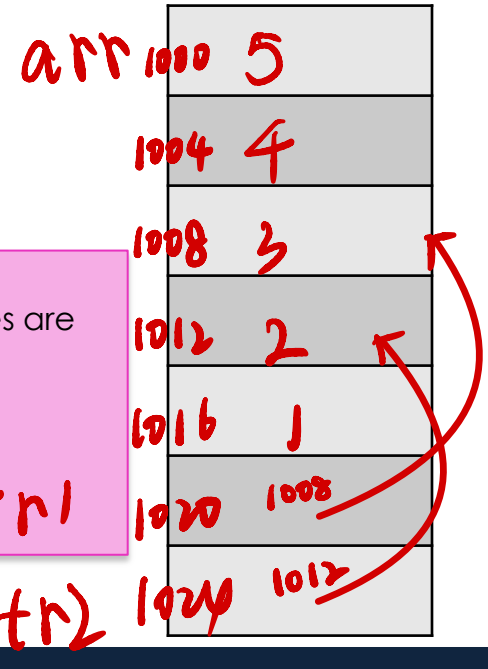
```
int arr[5] = { 5, 4, 3, 2, 1 };
int *ptr1 = arr + 2;
int *ptr2 = arr + 3;

cout << (ptr1 == ptr2) << endl;
cout << (ptr1 == ptr2 - 1) << endl;
cout << (ptr1 < ptr2) << endl;
cout << (*ptr1 < *ptr2) << endl;

++ptr1;
cout << (ptr1 == ptr2) << endl;
```

Poll

How many printed values are true?

A) 2
B) 3
C) 4
D) 5

Memory diagram



Array Indexing

- **Indexing** is a shorthand for **pointer arithmetic** followed by a **dereference**
  - `ptr[i]` is defined as `*(ptr+i)`
- Typically used with arrays:

```
int arr[4] = { 1, 2, 3, 4 };
cout << arr[3] << endl;
cout << *(arr + 3) << endl;
```

Equivalent

arr turns into a pointer

Indexing Exercise

- Which of the following are valid ways to access the element at index 3 from an array called `arr` (`select all that apply`)?

Poll:

A) `arr[3]`

B) `(*arr) + 3`

C) `*(&arr[0] + 3)`

D) `arr[2 + 1]`

E) `*(&arr[2] + 1)`

Functions and Array Parameters

- Arrays can be passed as parameters to functions
  - But this results in pointer decay!

the address of the first element of the array.

```
void func1(int arr[4]); // equivalent to int *arr
void func2(int arr[5]); // equivalent to int *arr
void func3(int arr[]);  // equivalent to int *arr
```

- No way of knowing how large an array that's passed in actually is, need to pass as extra argument

```
void func4(int arr[], int size);
```

so we need this.

Functions and Array Parameters

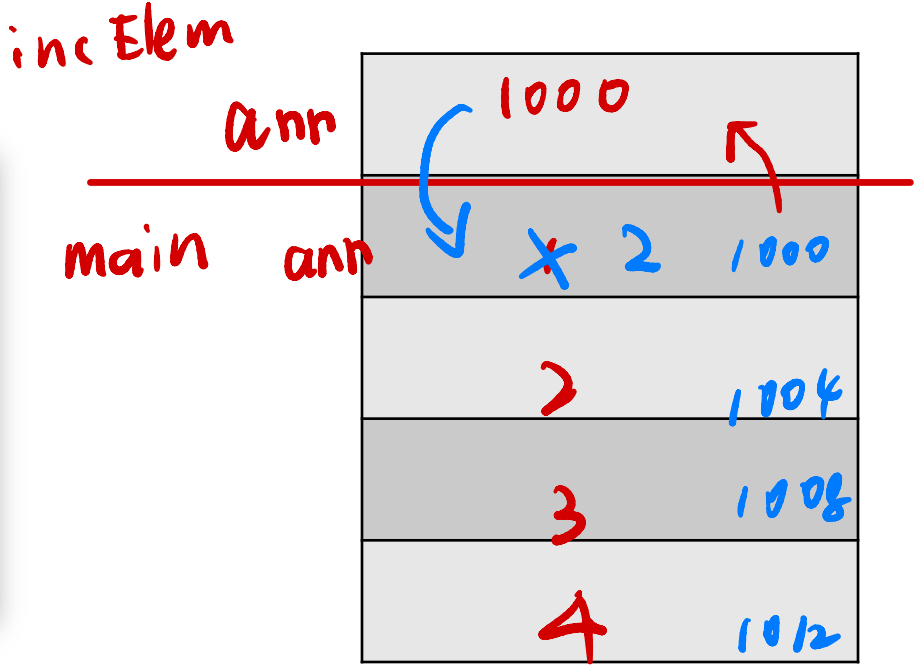
- "Yo teach, so this means arrays are passed by reference, right?"

- Passing an array as an arg **decays into a pointer**
- The pointer is passed by value (e.g. a copy of the address is made in the stack frame)
- Any modifications made to the array will be visible outside the function
  - It's ***as if*** the array was passed by ref

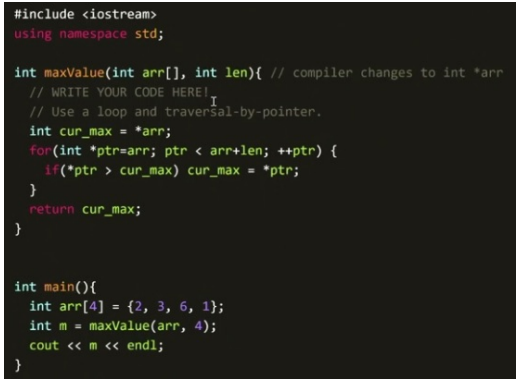
Functions and Array Parameters

```
void incElem(int arr[]) {
  arr[0]++;
}

int main() {
  int arr[4] = {1, 2, 3, 4};
  incElem(arr);
  cout << arr[0] << endl;
}
```

Memory diagram



Arrays Exercise



- Find the file "`L04.3_maxValue`" on Lobster.

  lobster.eecs.umich.edu

- Write the code for `maxValue (assume at least one element in array)`

```
int maxValue(int arr[], int len) {
  // WRITE YOUR CODE HERE!
  // Use a loop and indexing.
}

int main() {
  int arr[4] = {2, 3, 6, 1};
  int m = maxValue(arr, 4);
  cout << m << endl;
}
```

Agenda

- Finish up pointers
- Arrays
  - Array decay
- **Discussion on recent university news**

End of Lecture Material

- Let's take a break for a couple minutes
  - Feel free to head out if you'd like
- When we return, we can chat about the recent University news
- Frank discussion of sexual misconduct
- Feel free to post confidential questions / comments here: https://bit.ly/3ryuQCC

University News

- University President has been fired after an investigation revealed an inappropriate relationship with an employee
- This is the latest in a long chain of misconduct at UofM over the past few years
  - Former provost sexually harassed several women
  - 4 CS faculty have been accused of misconduct
    - 3 involving sexual misconduct in some form
- Clearly, this is indicative of a systemic issue
- How to trust the current systems when this behavior goes all the way to the top?

How to React?

- These events illustrate how important it is to speak up when we see something wrong
  - Me / 280 faculty, your lab instructor, RA
  - Anonymous dropbox
- If appropriate action isn't taken, tell someone else
- Personally, I'm very non-confrontative
  - I know that in order to act appropriately if/when I see something, I need to consciously commit to it now

Resources

- CSE information on Reporting Misconduct (guidelines for reporting concerns and misconduct, including anonymously, includes a **non-complete list** of “responsible employees" (i.e. mandatory reporters, must report any misconduct))
- Sexual Assault Prevention and Awareness Center (SAPAC) · **(**Support for survivors of sexual assault, 24/7 crisis line)
- U-M Counseling and Psychological Services (CAPS) (Provides info on counseling, 24/7 crisis line)
- ECRT (formerly Office of Institutional Equity)
- Other services: College of Engineering C.A.R.E. Center, Campus Mind Works, Depression Center, Services for Students with Disabilities, UHS, UM Psychiatric Emergency Services

Next Time

- Compound objects
  - How to create collections of objects of different types
- Lingering questions / feedback? I'll include an anonymous form at the end of every lecture: https://bit.ly/3oXr4Ah